Encoded scan : In the encoded scan, scan lines (SL_2-SL_0) are decoded externally to provide 8 scan lines. We know that 8279 provides 8 returns lines. Therefore, the maximum size of keyboard matrix is $8 \times 8 = 64$. When the key is pressed, 8279 stores the encoded status of scan lines and return lines along with the status of SHIFT and CNTL/STB keys into the FIFO RAM, as shown in the Fig. 10.18.

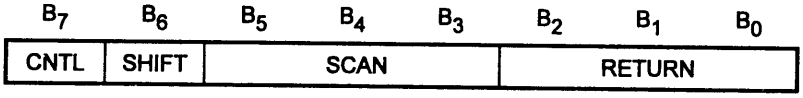


Fig. 10.18 Scanned keyboard data format

CNTL is the MSB of the character and shift is the next most significant bit. The next three bits are from the scan counter. The last three bits indicate to which return line the key is connected. With this 8-bit key code 8279 can recognize 256 (2^8) different characters.

Example 10.3 : *Find the key code for condition given below : CNTL/STB SHIFT keys are open.The pressed key is connected to scan line 2 and return line 4.*

Solution :

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	1	0	1	0	1	0	0

CNTL = 1

SHIFT = 1

Scan Code = 0 1 0 (scan line 2)

Return Code = 1 0 0 (return line 4)

$\therefore$ Key code = D4H

Decoded scan :

In the decoded mode, the internal decoder decodes the least significant 2-bits of scan lines internally to provide four possible combinations on the scan lines ($SC_3 - SC_0$) : 1110, 1101, 1011 and 0111. Therefore maximum size of keyboard matrix is $8 \times 4 = 32$. In this mode, keycode is generated in similar way as in the encoded mode, only bit 5 of keycode is always 0. Therefore, 8279 can recognize only 128 (2^7) characters.

The scanned keyboard mode allows key depressions in 2-key lockout or N-key rollover mode with key debounce.

2-key lockout : In this mode, simultaneous key depression is not allowed. When any key is depressed, the debounce logic is set and 8279 checks for any other key depress

during next two scans. Now we will see how this mode reacts with three possible conditions that can occur during debounce scanning.

Condition 1 : If other key depress not found during next two scans, it is a single key depression and the key code is entered into FIFO RAM along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO RAM. If the FIFO RAM was full, the key will not be entered and the error flag will be set.

Condition 2 : If another key depress is encountered, no entry to the FIFO can occur. If all other keys are released before first one, then it will be entered to the FIFO. If first key is released before any other, it will be entirely ignored.

Condition 3 : If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

N-Key rollover : In N-Key rollover, each key depression is treated independently from all others. When a key is depressed, the debounce logic is set and 8279 checks for key depress during next two scans. Now we will see how this mode reacts with three possible conditions that can occur during debounce scanning.

Condition 1 : If key is still pressed then key is entered into the FIFO.

Condition 2 : If other keys are pressed, they are recognized and entered into the FIFO.

Condition 3 : If a simultaneous depression occurs, the keys are recognized and entered according to the keyboard scan found them.

Scanned sensor matrix : In the sensor matrix mode, image of the sensor matrix is kept in the sensor RAM. The status of the sensor switches are input directly to the sensor RAM. 8279 scans rows one by one and stores the status of each row in the corresponding location in the sensor RAM. For example, when 8279 scans first row of sensor matrix it stores the status of first row in the location 0 of the sensor RAM. At the end of sensor matrix scan if any sensor value change is detected then 8279 sets 'S' bit in the status register and activates the IRQ signal. In the autoincrement mode, the IRQ line is cleared by issuing End of Interrupt command, otherwise it is cleared by the first data read operation. When multiple changes in the sensor matrix occurs, multiple interrupts are generated. In sensor matrix mode, the debounce logic is inhibited. Although it is inhibited, sensor matrix mode has the advantage that CPU knows how long the sensor was closed and when it was released. The scanned keyboard mode can only indicate validated key closure. In encoded mode, size of sensor matrix is 8×8 whereas in decoded mode size of sensor matrix is 8×4. In both the modes CNTL and SHIFT lines are ignored.

Strobed input mode

In the strobed input mode, data is entered to the FIFO RAM from the returned lines. The data is entered at the rising edge of the CNTL/STB signal.

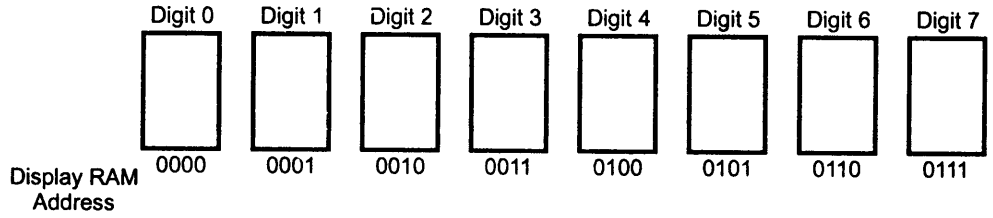
10.5.4.2 Display Modes

The 8279 provides 2 basic output modes

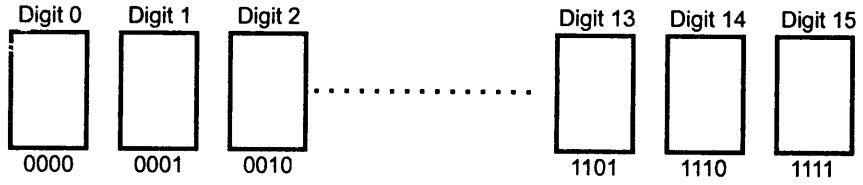
- Left entry (Typewriter type)
- Right entry (Calculator type)

Left Entry

In the left entry mode, 8279 displays characters from left to right in the multiplexed displays like a typewriter. In this, each display position is directly corresponds to a byte (or nibble) in the display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8-character display) is the right most display character, as shown in the Fig. 10.19.



(a) 8-character display left entry mode



(b) 16-character display left entry mode

Fig. 10.19

Entering characters from possible zero causes the display to fill from the left. The 17^{th} (9^{th} in case 8 character display) character is entered back in the left most position and filling again proceeds from there, as shown in the Fig. 10.20. The characters can be displayed on the specific digit by loading character code in the corresponding location in the display RAM.

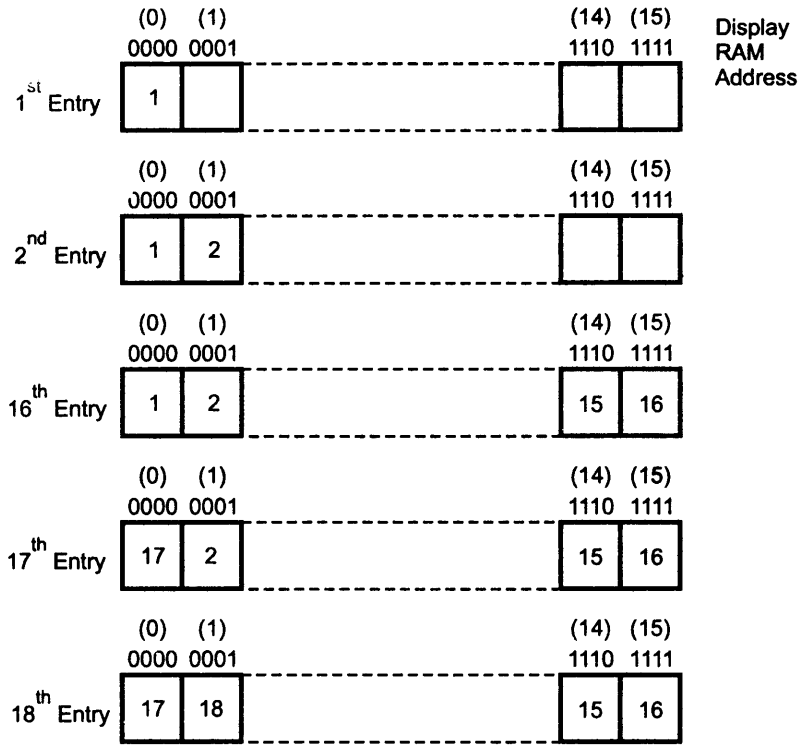


Fig. 10.20 16-characters left entry mode

Autoincrement in Left Entry

In left entry mode, if autoincrement flag is set to 1 after each write operation display RAM address is incremented by one so that it will point the next location in the display RAM. This autoincrement facility allows user to load display RAM in a sequential manner, and it is not necessary to specify display RAM address for each write operation.

Right Entry

In the right entry mode, 8279 displays characters from right to left in the multiplexed display like a calculator. The first entry is displayed on the right most display. The next entry is also displayed on the right most display after the display is shifted left one character, as shown in the Fig. 10.21.

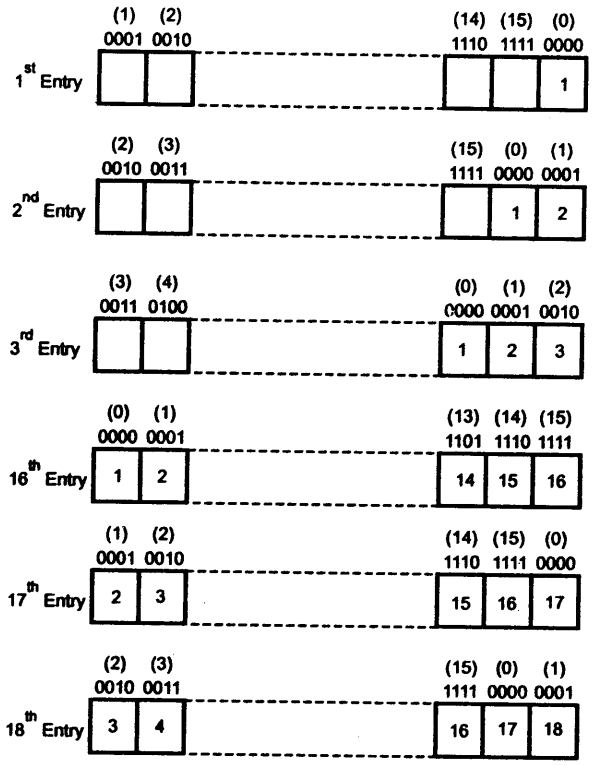


Fig. 10.21 16-characters right entry mode with autoincrement

When new character is entered, it shifts all previous characters left by one position and displays new entry on the right most display.

Autoincrement in Right Entry

In the right entry mode, autoincrementing and non-autoincrementing have the same effect as in the left entry except if the address sequence is changed. Some examples with changed address sequence.

First character is displayed on the right most digit of the display. After second entry first character is shifted left and second character is displayed on the right most digit of the display. In the third entry, address of display RAM is changed to 5, displaying the third character at fifth digit after shifting the previous characters 1 digit left. In the 4^{th} Entry the new character is displayed at 5^{th} digit after shifting all previous characters 1 digit left, and this sequence is continued. (Fig. 10.22 see on next page).

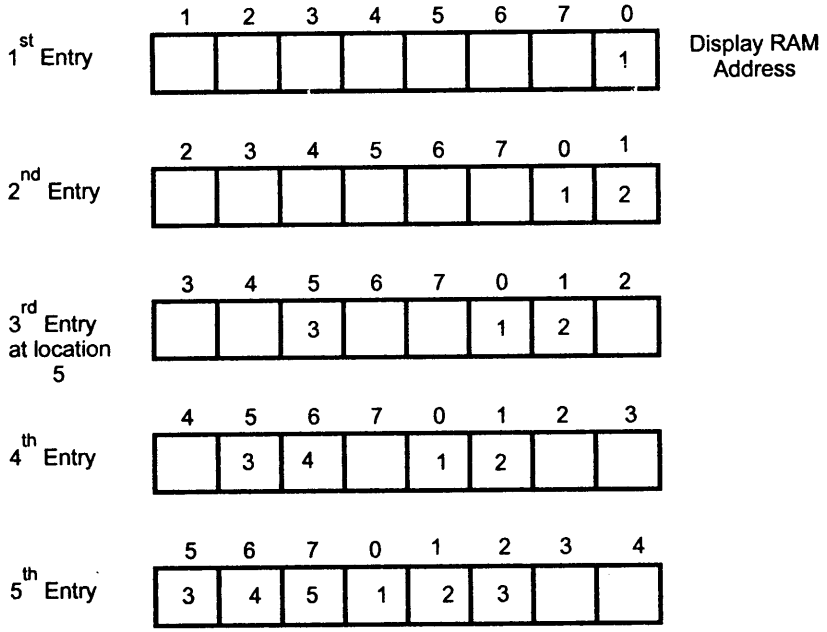


Fig. 10.22 8-characters right entry with autoincrement

10.5.5 8279 Commands

In the last sections we have seen various operating modes of 8279. To program 8279 in the desired mode it provides eight command words. The command words are sent on the data bus with $\overline{CS}$ low and A_0 high and are loaded to the 8279 on the rising edge of $\overline{WR}$. 8279 differentiate these commands by checking 3 most significant bits of the command word.

10.5.5.1 Keyboard/Display Mode Set Command (000)

This command is used to program operating modes of keyboard and display. Three least significant bits decide the keyboard mode and next two bits decide the display mode, as shown in the tables.

Command word format

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	D	D	K	K	K

Fig. 10.23 Keyboard/display command word format

K	K	K	Keyboard modes
0	0	0	Encoded Scan Keyboard-2 key lockout
0	0	1	Decoded Scan Keyboard-2 key lockout
0	1	0	Encoded Scan Keyboard-N key rollover
0	1	1	Decoded Scan Keyboard-N key rollover
1	0	0	Encoded Scan Sensor Matrix
1	0	1	Decoded Scan Sensor Matrix
1	1	0	Strobed Input, Encoded Display Scan
1	1	1	Strobed Input, Decoded Display Scan

Table 10.5 Keyboard modes

D	D	Display modes
0	0	8 8-bit character display - Left Entry
0	1	16 8-bit character display - Left Entry
1	0	8 8-bit character display - Right Entry
1	1	16 8-bit character display - Right Entry

Table 10.6 Display modes

Example 10.4 : *Give the command word to set keyboard/Display mode for the following configuraton.*
Encoded scan keyboard - N key rollover
16 8-bit character display - Right Entry.

Solution : Command word

0	0	0	D	D	K	K	K
0	0	0	1	1	0	1	0

= 1AH

Example 10.5 : *The microprocessor system has a configuration given below. Find the keyboard/display command word.*
8 × 4 matrix keyboard - 2 key lockout
4 Digit 7-segment display left entry.

Solution : The system has 8 × 4 matrix keyboard and 4 digit display. Hence, only 4 scan lines are sufficient. The decoded mode of 8279 provides 4 scan lines directly and these lines can be used directly to interface 8 × 4 matrix keyboard and 4 digit display without external decoder. Therefore, we should select keyboard and display modes as:

Keyboard mode : Decoded scan keyboard - 2 key lockout

Display mode : 8-bit character display left entry

Command word :

0	0	0	D	D	K	K	K
0	0	0	0	0	0	0	1

= 01H

10.5.5.2 Program Clock Command (001)

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock by a programmable integer value given in the program clock command word, to generate internal frequency Fig. 10.24 shows format for program clock command word.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	1	P	P	P	P	P

Fig. 10.24 Program clock command word format

Bits PPPPP determine the value of this integer which ranges from 2 to 31. To give proper scan and key debounce times the internal frequency should be 100 kHz. Therefore, prescaler integer value should be selected to get 100 kHz internal frequency.

$$\text{Prescaler value} = \frac{\text{External clock}}{100\ \text{kHz}}$$

Example 10.6 : *Find the program clock command word if external clock frequency is 2 MHz.*

Solution :

$$\text{Prescaler value} = \frac{2\times10^6}{100\times10^3} = 20 = (10100)_2$$

$\therefore$ Command word $= (00110100)_2 = 34H$

10.5.5.3 Read FIFO/Sensor RAM Command (010)

To read data from FIFO/sensor RAM, it is necessary to set 8279 in read FIFO/sensor RAM mode. Read FIFO/sensor RAM command is used for this purpose Fig. 10.25 shows the format for Read FIFO/Sensor RAM command.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	0	AI	X	A	A	A

Fig. 10.25 Read FIFO/sensor RAM command word format

Here, three least significant bits, (AAA) specify the address of the sensor RAM and bit B_4, if 1 enables autoincrement mode. In the scan keyboard mode, the autoincrement flag (AI) and the FIFO RAM address bits (AAA) are irrelevant. In this mode, 8279 provides data for each subsequent read in the same sequence in which the data first entered in the FIFO RAM.

In the sensor matrix mode, the sensor RAM address bits AAA select one of the 8 rows of the sensor RAM. If the autoincrement flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Example 10.7 : *Write a command word to read data from FIFO RAM.*

Solution : We know that, in scan keyboard mode, the autoincrement flag (AI) and the FIFO RAM address bits (AAA) are irrelevant. Therefore, command word to read data from FIFO RAM is as given below.

			AI	D	A	A	A	
0	1	0	X	X	X	X	X	= 40H

Note : X = Don't care. Taking don't cares equal to zeros we get command word to read data from FIFO RAM is 40H.

Example 10.8 : *Write a command word to read third location with autoincrement of the sensor RAM in sensor matrix mode.*

Solution : Command word :

			AI		A	A	A	
0	1	0	1	X	0	1	1	= 53H

10.5.5.4 Read Display RAM Command (011)

To read data from display RAM, it is necessary to set 8279 in read display RAM mode. Read display RAM command is used for this purpose. Fig. 10.26 shows the format for Read Display RAM command

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	1	AI	A	A	A	A

Fig. 10.26 Read display RAM command word format

Here, four least significant bits (AAAA) specify the address of the 16 byte display RAM and bit B_4, if 1, enables autoincrement mode. If the bit B_4 (AI) is set, display RAM address is incremented after each read command to display RAM.

Example 10.9 : *Write a command word to read fourth location with autoincrement of the display RAM.*

Solution : Command word

			AI	A	A	A	A	
0	1	1	1	0	1	0	0	= 74H

10.5.5.5 Write Display RAM Command (100)

To write data into display RAM, it is necessary to set 8279 in write display RAM mode. Write display RAM command is used for this purpose. Fig. 10.27 shows the format for write Display RAM command.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	0	0	AI	A	A	A	A

Fig. 10.27 Write display RAM command word format

Here, four least significant bits (AAAA) specify the address of the 16 byte display RAM and bit B_4, if 1, enables autoincrement mode. If the bit B_4 (AI) is set, display RAM address is incremented after each write command to display RAM.

➠ **Example 10.10 :** *Write a command word to write fifth location without autoincrement of the display RAM.*

Solution : Command word

			AI	A	A	A	A	
1	0	0	0	0	1	0	1	= 85H

10.5.5.6 Display Write Inhibit/Blanking Command (101)

We know that, display RAM data is sent on the two 4-bit ports (B_3-B_0 and A_3-A_0). This two 4-bit ports can be individually inhibited or blanked with display write inhibit/blanking command. Fig. 10.28 shows the format for display write Inhibit/Blanking Command

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_1
1	0	1	X	IW A	IW B	BL A	BL B

Fig. 10.28 Display write Inhibit/Blanking command word format

The IW bits are used to mask nibble A (4-bit port A) and nibble B (4-bit port B) in applications requiring separate 4-bit display ports. By setting the IW flag (I/W = 1) for one of the ports, the port can be masked so that entries to the display RAM from the CPU donot affect other port.

The BL bits are used to blank the individual nibbles. This command loads the blank code (All zeros, 20H, or All ones) determined by the last issued clear command, in the display RAM to blank the display.

Note : After reset blank code is set to all zeros.

Example 10.11: *Write a command word to inhibit nibble A of the display.*

Solution : Command word

				IW A	IW B	BL A	BL B	
1	0	1	X	1	0	0	0	= A8H

10.5.5.7 Clear Command (110)

Clear command is used to clear all the rows of the display RAM with a selectable blanking code, to clear status of FIFO RAM and to reset interrupt output line. Fig. 10.29 shows the format of display command.

1	1	0	CD_2	CD_1	CD_0	CF	CA

Fig. 10.29 Clear command word format

CD bits (CD_0-CD_1) are used to select the blanking code as given below.

CD_1	CD_0	Blanking code
0	X	All Zeros (for common cathode displays)
1	0	20H (for alphanumeric displays)
1	1	All ones (for common anode displays)

Bit CD_2, when set to one, enables clear display

Bit CF, when set to one, clears the status of the FIFO, resets the interrupt output line and sets the sensor RAM address to 000.

CA, the clear all bit, has the combined effect of CD and CF; it uses the CD clearing code on the displays RAM and also clears FIFO status. It also resynchronizes the internal timing chain.

Example 10.12 : *Write a command word to set blanking code for common anode display and to clear the FIFO status.*

Solution : Blanking code for common anode display is all ones and which can be set by writing CD_1 = 1 and CD_2 .

			CD_2	CD_1	CD_0	CF	CA	
1	1	0	1	1	1	1	0	= DEH

OR

			CD_2	CD_1	CD_0	CF	CA	
1	1	0	X	1	1	X	1	= CDH

10.5.5.8 End Interrupt/Error Mode Set Command (111)

In the sensor matrix mode, if any change in sensor value is detected, IRQ line goes high at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the autoincrement flag is set to zero. But if autoincrement flag is set to one then it is necessary to issue End Interrupt Command to clear the IRQ line. Fig. 10.30 shows the format for End interrupt/Error mode set command.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	1	1	E	X	X	X	X

Fig. 10.30 End interrupt/Error mode set command word format

For the N key rollover mode, if the E bit is programmed to '1', the 8279 will operate in the Special Error Mode. In the special error mode, if two keys are depressed during single debounce, the error flag in the FIFO status word is set.

Example 10.13 : *Write a command word to clear IRQ line in a sensor matrix mode.*

Solution : Command word :

			E					
1	1	1	X	X	X	X	X	= E0H

Note : Status of E bit can be 0 or 1

FIFO STATUS REGISTER

It is used in the keyboard and strobed input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. Fig. 10.31 shows the format of FIFO status word.

As shown in the Fig. 10.31, there are two types of possible errors : Overrun and underrun. Overrun error occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

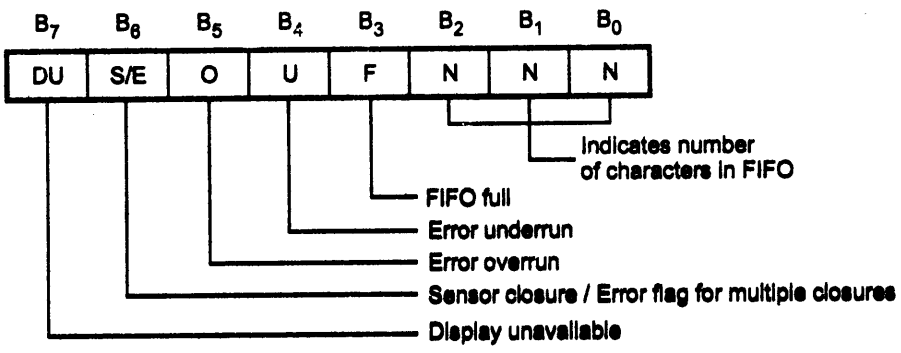


Fig. 10.31 FIFO status word

During clear display or clear all command, display RAM is not available for user. This is indicated DU bit in the FIFO status register.

In the sensor matrix mode, a S/E bit is set in the FIFO status word to indicate that atleast one sensor closure indication is contained in the sensor RAM.

In the special error mode, a S/E bit is set in the FIFO status word to indicate that a simultaneous multiple closure error has occurred.

10.5.6 Interfacing 8279 in I/O Mapped I/O

Fig. 10.32 shows the interfacing of 8279 with 8086 in I/O mapped I/O technique. Here $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is low, indicating I/O bus cycle. Reset out signal from 8086 system is connected to the Reset signal of the 8279. CLK input of 8279 is driven from the clock signal of 8086 system. A_1 signal from the 8086 is connected to the A_0 input of 8279. The chip select signal, $\overline{CS}$ is generated using decoding circuit. Interrupt signal from the 8279 is connected to the interrupt input of 8086.

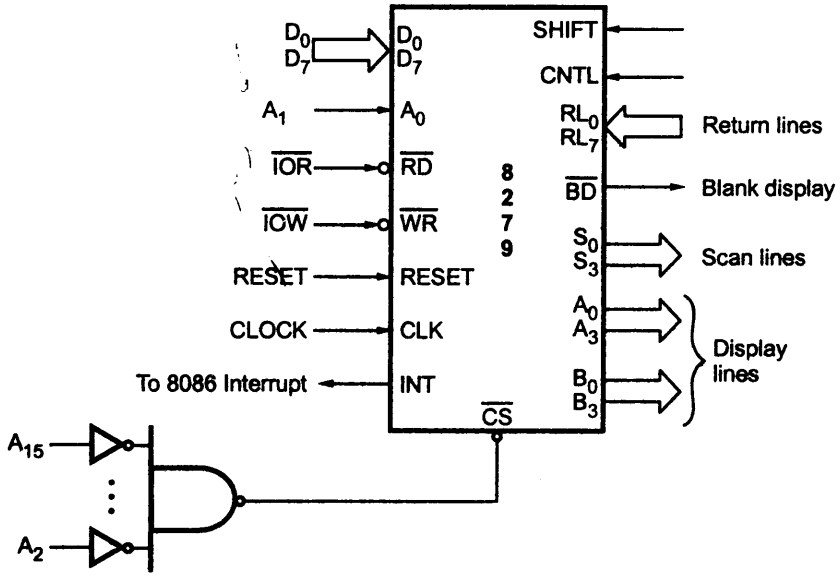


Fig. 10.32 Interfacing of 8279 in I/O mapped I/O

I/O Map :

Data/Control Register	Address lines																Address
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Data Register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00H
Control Register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	02H

10.5.7 Interfacing 8279 in Memory Mapped I/O

Fig. 10.33 shows the interfacing of 8279 with 8086 in memory mapped I/O technique. Here, $\overline{RD}$ and $\overline{WR}$ signals are activated when M/$\overline{IO}$ signal is high, indicating memory bus cycle. To get absolute address, all remaining address lines (A_2-A_{19}) are used to decode the address for 8279. Other signal connections are same as in I/O mapped I/O.

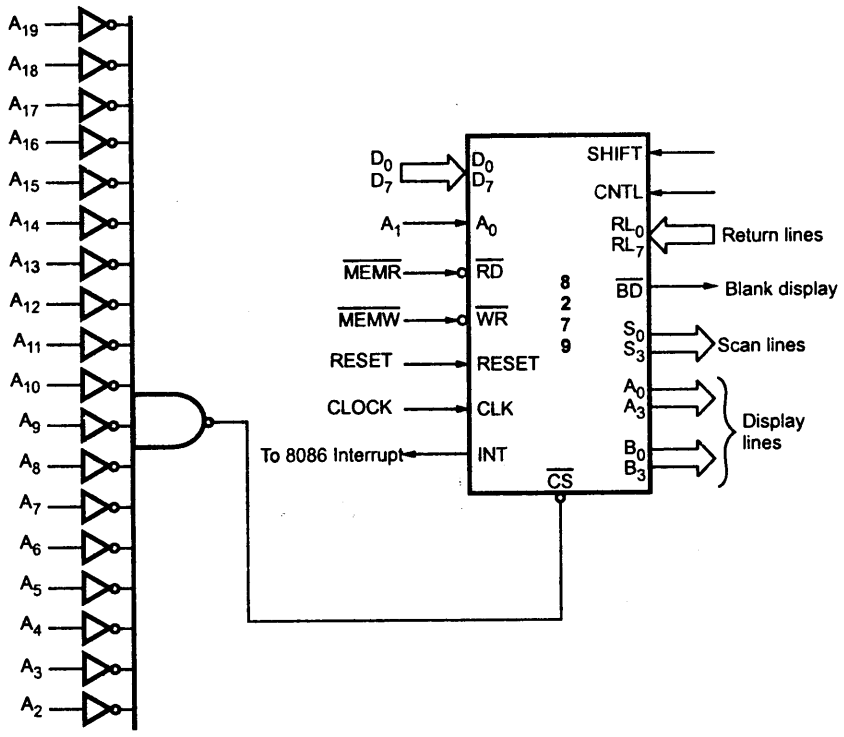


Fig. 10.33 Interfacing of 8279 in memory mapped I/O

I/O Map :

Data/ Control Register	Address lines																				Address
	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Data Register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00H
Control Register	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	02H

10.5.8 Applications

In this section we will discuss many useful applications with different modes of keyboard and display interfacing. In addition to this we are going to see the software requirement to control the interfacing circuits. All these applications are illustrated using different examples.

Program 30 : Hardware and software for 8 × 8 keyboard interface using 8279

Program statement : Interface an 8 × 8 matrix keyboard to 8086 through 8279 in 2-key lockout mode and write an assembly language program to read keycode of the pressed key. The external clock frequency is 2 MHz. Use I/O mapped I/O technique.

Solution : The 8 × 8 matrix keyboard can be interfaced to 8086 through 8279 in two ways.

1. Without interrupt signal
2. With interrupt signal (Interrupt driven input)

We will see both the ways one by one.

1. Without interrupt signal

Hardware : Fig. 10.34 shows the interfacing of 8 × 8 matrix keyboard.

(See Fig. 10.34 on next page.)

I/O Map

Data/Control Register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data Register	1	0	0	0	0	0	0	0	80 H
Control Register	1	0	0	0	0	0	1	0	82 H

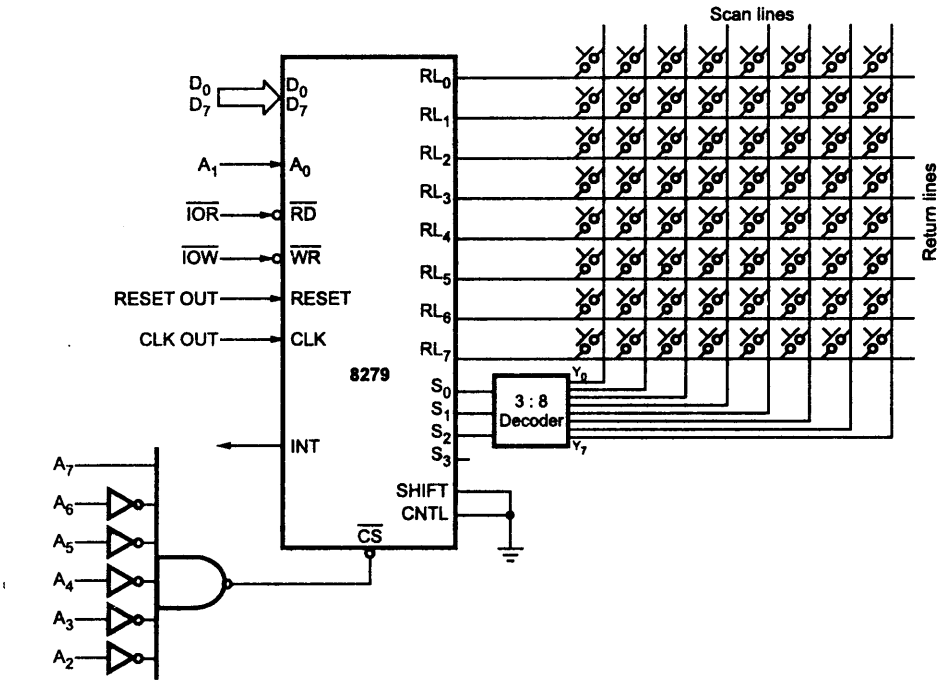


Fig. 10.34 Interfacing of 8 × 8 matrix keyboard

Software :

Step 1 : Find keyboard/display command word. To interface 8 × 8 matrix keyboard we need 8 scan lines and 8 return lines. To get 8 scan lines. We have to select encoded scan keyboard mode. Therefore, the keyboard/display command word for above keyboard configuration is given as follows :

0	0	0	D	D	K	K	K	
0	0	0	X	X	0	0	0	= 00H

Note : 000 → Encoded scan keyboard - 2 key lockout

X → don't care

Step 2 : Find program clock command word

External clock frequency is 2 MHz

$$\therefore \quad \text{Prescaler value} = \frac{2\text{MHz}}{100\text{kHz}}$$

$$= 20 = (10100)_2$$

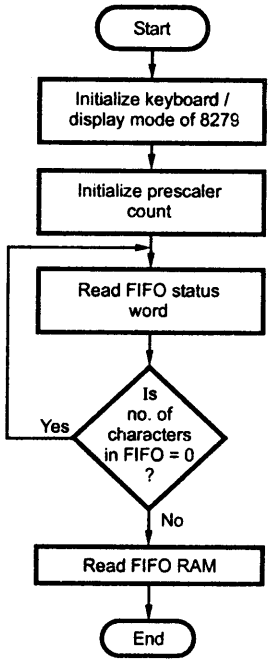
Therefore, the program clock command word is as given below :

			P	P	P	P	P	
0	0	1	1	0	1	0	0	= 34H

Step 3 : Find Read FIFO/sensor RAM command word we want to read first entry from the FIFO RAM. Therefore command word is as given below.

			AI		A	A	A	
0	1	0	0	X	0	0	0	= 40H

Flowchart :



Program :

```
                MOV AL,  00H
                OUT 81H, AL   ; Initialize keyboard/display
                              ; in encoded scan keyboard-2 keylockout mode
                MOV AL,  34H
                OUT 81H, AL   ; Initialize prescaler count
BACK :          IN AL,   81H  ; Read FIFO status word
                AND AL,  07H  ; Mask bit B3 to B7
                JZ BACK       ; If 0, key is not pressed wait for key
                              ; press
                              ; otherwise read FIFO RAM
                MOV AL, 40H   ; Initialize 8279 in read
                OUT 81H, AL   ; FIFO RAM mode
                IN AL,   80H  ; Read FIFO RAM (keycode)
```

2. With interrupt signal

Hardware :

Fig. 10.35 shows the interfacing of 8×8 matrix keyboard in interrupt driven keyboard mode.

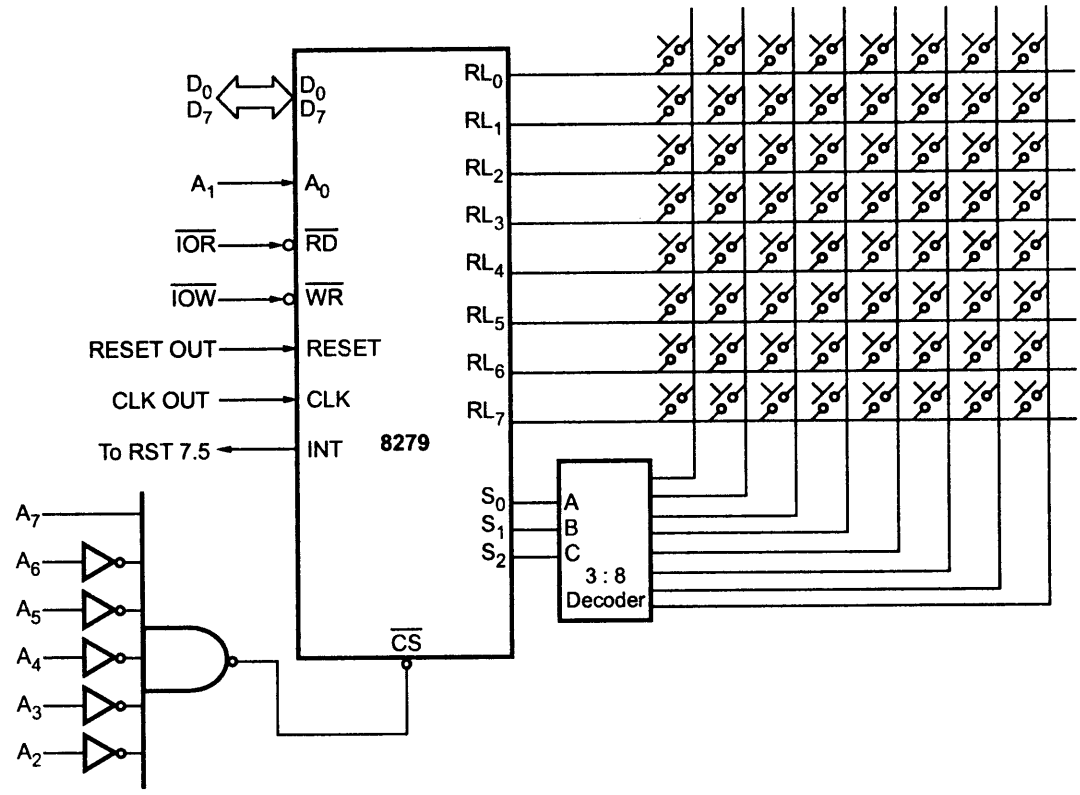


Fig. 10.35 Interfacing of 8×8 matrix keyboard in interrupt driven keyboard mode

In the interrupt driven mode interrupt line from 8279 is connected to the INTR the interrupt input of 8086.

Software : All the command words required to initialize 8279 are same as in the noninterrupt mode.

Flowchart :

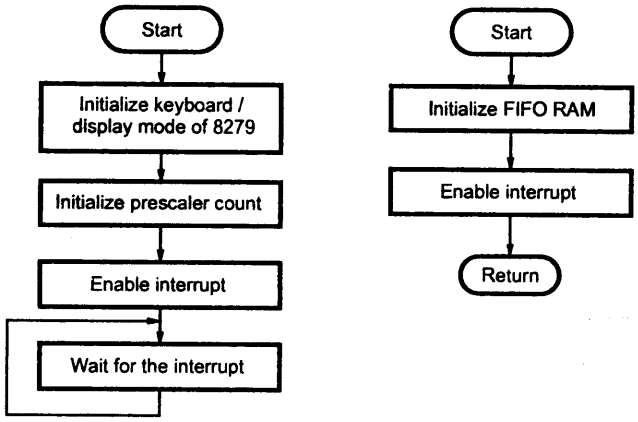


Fig. 10.36

Main program

```
        MOV AL, 00H
        OUT 82H,AL    ; Initialize keyboard/display in encoded
                      ; scan keyboard 2 key lockout mode
        MOV AL, 34H
        OUT 82H,AL    ; Initialize prescaler count
HERE :  JMP HERE      ; Wait for the interrupt
```

Interrupt subroutine

```
        MOV AL, 40H   ; Initialize 8279 in read FIFO
        OUT 82H, AL   ; RAM mode
        IN AL, 80H    ; Read FIFO RAM (keycode)
        RET           ; Return to main program
```

In the interrupt driven keyboard, when key is pressed, key code is loaded into FIFO RAM and interrupt is generated. This interrupt signal is used to tell CPU that there is a keycode in the FIFO RAM. CPU then initiates read command with in the interrupt service routine to read keycode from the FIFO RAM.

Program 31 : Hardware and software to interface 8×4 matrix keyboard using 8279

Program statement : Interface an 8 × 4 matrix keyboard to 8086 through 8279.

Solution : Fig. 10.37 shows interfacing of an 8 × 4 matrix keyboard to 8086 through 8279.

(See Fig. 10.37 on next page.)

As keyboard is having 8 rows and 4 columns, only 4 scan lines are required and we can avoid external decoder to generate scan lines by selecting decoded scan keyboard mode.

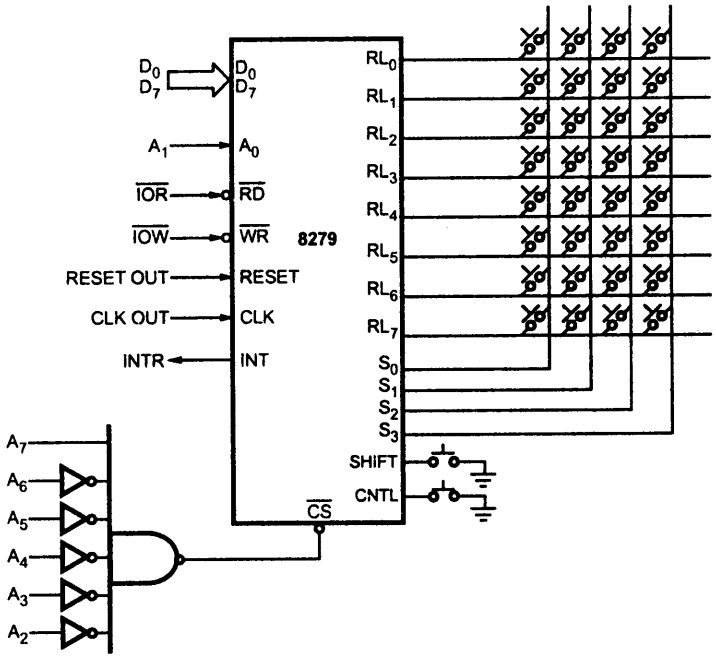


Fig. 10.37 Interfacing 8 × 4 keyboard matrix in decoded scan keyboard mode

Main program

```
        MOV AL, 00H
        OUT 82H, AL        ; Initialize keyboard/display in encoded
                           ; scan keyboard 2 key lockout mode
        MOV AL, 34H
        OUT 82H, AL        ; Initialize prescaler count
HERE :  JMP HERE           ; Wait for the interrupt
```

Interrupt subroutine

```
        MOV AL, 40H        ; Initialize 8279 in read FIFO
        OUT 82H, AL        ; RAM mode
        IN AL, 80H         ; Read FIFO RAM (keycode)
        RET                ; Return to main program
```

Program 32 : Hardware and software to interface eight 7-segment digits using 8279

Program statement : Interface 8/7-segment digits (common cathode) to 8086 through 8279 and write an 8086 assembly language program to display 1 to 8 on the eight seven segment digits. External clock frequency is 3 MHz.

Solution : Fig. 10.38 (see Fig. 10.38 on next page) shows the interfacing of eight 7-segment digits to 8086 through 8279.

As shown in the Fig. 10.38, eight display lines (B_0- B_3 and A_0-A_3) are buffered with the help of transistor and used to drive display digits. These buffered lines are connected in parallel to all display digits. S_0, S_1 and S_2 lines are decoded and decoded lines are used for selection of one of the eight digits.

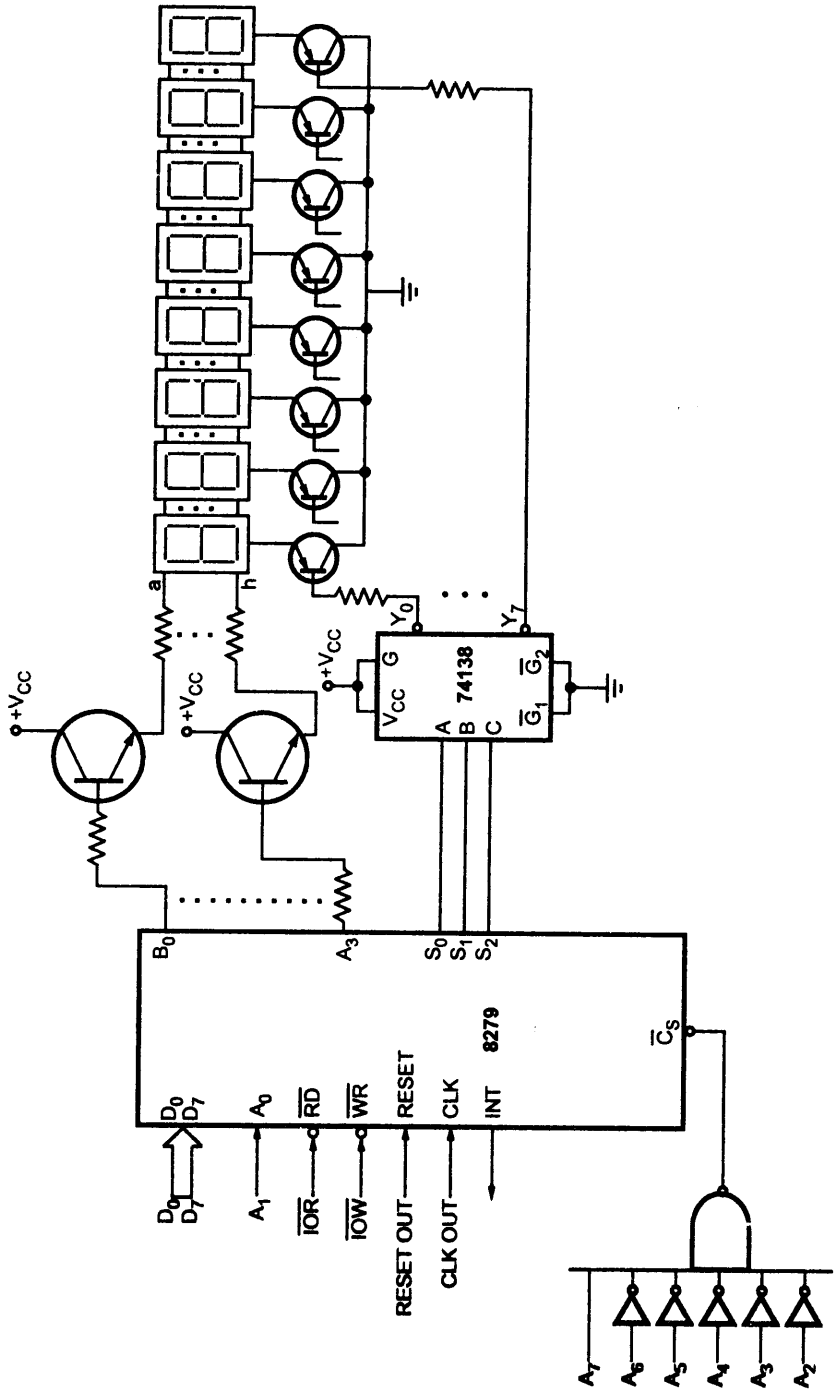

a
h
+VCC
+VCC
+VCC
Y0
Y7
G
VCC
74138
G2
G1
A
B
C
B0
A3
S0
S1
S2
8279
CS
D0
D7
A0
RD
WR
RESET
CLK
INT
D0
D7
A1
IOR
IOW
RESET OUT
CLK OUT
A7
A6
A5
A4
A3
A2

Fig. 10.38

Software : To display 1 to 8 numbers on the eight 7-segment digits we have to load 7-segment codes for 1 to 8 numbers in the corresponding display locations.

Number	h	g	f	e	d	c	b	a	Code
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F

Table 10.7 7-Segment codes for common cathode display

Step 1 : Find keyboard/display command word. To interface 8 digit 7-segment display we need 8/8-bit character display mode with left entry. For selection of 8 digits we need encoded scan mode. Therefore, the keyboard/display command word is as given below.

			D	D	K	K	K	
0	0	0	0	0	0	0	0	= 00H

Step 2 : Find program clock command word. External clock frequency is 3 MHz.

$$\therefore \quad \text{Prescaler value} = \frac{3\,\text{MHz}}{100\,\text{MHz}} = 30 = (11110)_2$$

Therefore, the program clock command word is as given below.

			P	P	P	P	P	
0	0	1	1	1	1	1	0	= 3EH

Step 3 : Find write display RAM command word. We want to write first eight locations of display RAM with corresponding 7-segment codes. So we start from first location with autoincrement mode by command word given below.

			AI	A_3	A_2	A_1	A_0	
1	0	0	1	0	0	0	0	= 90H

Flowchart :

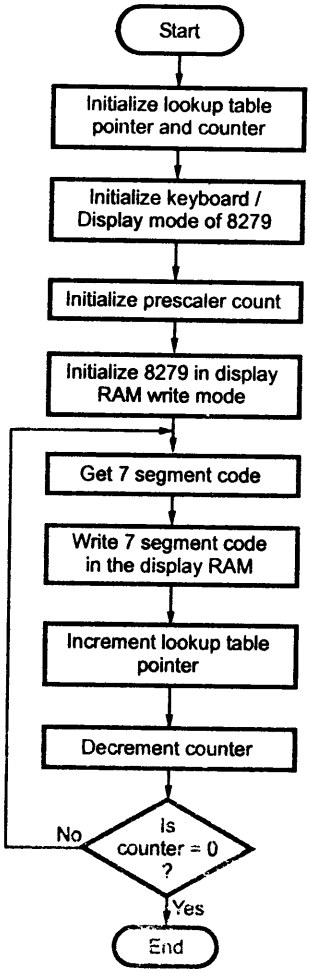


Fig. 10.39

Program :

```
        MOV SI, LOOK_UP_TABLE ; Initialize lookup table pointer
        MOVCL, 08H            ; Initialize counter
        MOV AL, 00H           ; Initialize keyboard/display
        OUT 82H, AL           ; Mode
        MOV, AL, 3EH          ; Initialize prescaler count
        OUT 82H, AL
        MOV, AL, 90H          ; Initialize 8279 in write Display
        OUT 82H, AL           ; RAM mode
BACK :  MOV AL, [SI]          ; Get the 7-segment code
        OUT 80H, AL           ; Write 7-segment code in display
                              ; RAM
```

```
        INC SI              ; Increment lookup table pointer
        DCR CL              ; Decrement counter
        JNZ BACK            ; if count = 0 stop otherwise go
                            ; to back
LOOK_UP-TABLE dB 66H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH
```

Review Questions

1. *What do you mean by static display and multiplexed display ? Draw the circuit arrangement for interfacing 4 digit multiplexed display to 8086 with the help of 8255.*
2. *What is disadvantage of software approach used for interfacing keyboard and display ?*
3. *List the features of 8279.*
4. *Draw the functional pin diagram of 8279 and explain the function of different pins.*
5. *Draw and explain the internal block diagram of 8279.*
6. *What do you mean by encoded scan and decoded scan ?*
7. *Explain the input modes provided by 8279.*
8. *Explain the terms 2-key lockout and N-key rollover.*
9. *Why maximum size of keyboard matrix is* $8 \times 8 = 64$, *when interfaced with 8279?*
10. *Explain the display modes provided by 8279.*
11. *Interface a* 4×4 *matrix keyboard to the microprocessor using 8279 IC. Discuss the operation.*
12. *Interface* 4×4 *matrix keyboard and 6 displays to the 8086 system using 8279 IC. Write initialisation program for encoded key scan and left entry for display.*
13. *Interface a 16 keys keyboard and four 7-segment LED's to 8086 using 8279. Write a program to read the keyboard and store the key read in the location KEYBUF.*

❑❑❑

11 8086/8088 Based Multiprocessing Systems

11.1 Introduction

If a microprocessor system contains two or more components that can execute instructions independently, then the system is called **multiprocessor system.** Multiprocessor system uses a distributed approach. Here, more than one processors are used to do the subtasks instead of doing entired task by a single processor. This system has following advantages :

1. Improves cost/performance ratio of the system.
2. Several processors may be combined to fit the needs of an application while avoiding the expense of the unneeded capabilities of a centralized system. Yet this system provides room for expansion.
3. Tasks are divided among the modules. If failure occurs, it is easier and cheaper to find and replace the malfunctioning processor than replacing the failing part of complex processor.

Thus we can say that, multiprocessor systems aim to improve throughput, reliability, flexibility and availability. The Fig. 11.1 shows typical multiprocessor organization. It consists of two or more processors of approximately comparable capabilities. All processor share access to common sets of memory modules, input-output channels, and peripheral devices. Along with shared memory and input-output devices, each processor has its own local memory and input-output devices. The interprocessor communication is done through the shared memories or through an interrupt network. Most importantly, the entire system is controlled by the single operating system providing interactions between processors and their programs at various levels.

The multiprocessor systems are implemented using one of the two basic architectures : loosely coupled architecture and closely coupled architecture. The systems using these architectures are known as loosely coupled systems and closely coupled systems respectively.

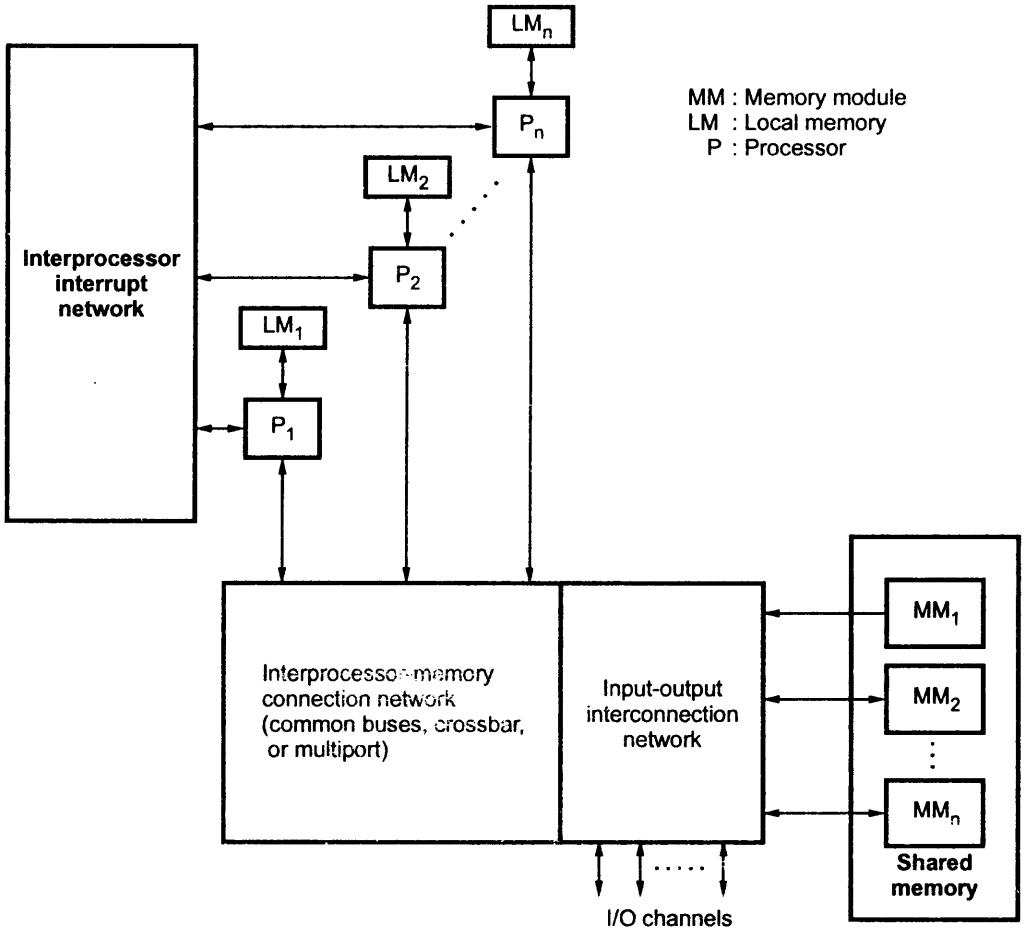


Fig. 11.1 Function design of typical multiprocessor system

11.2 Closely Coupled System using 8086

The Fig. 11.2 shows the simplest from of closely coupled configuration. In this configuration, the CPU (8086) is the master or host and the supporting processor is the slave. Therefore, two 8086s cannot appear in this configuration. The CPU provides the bus control logic. So the bus request signal from the supporting processor is connected to the CPU.

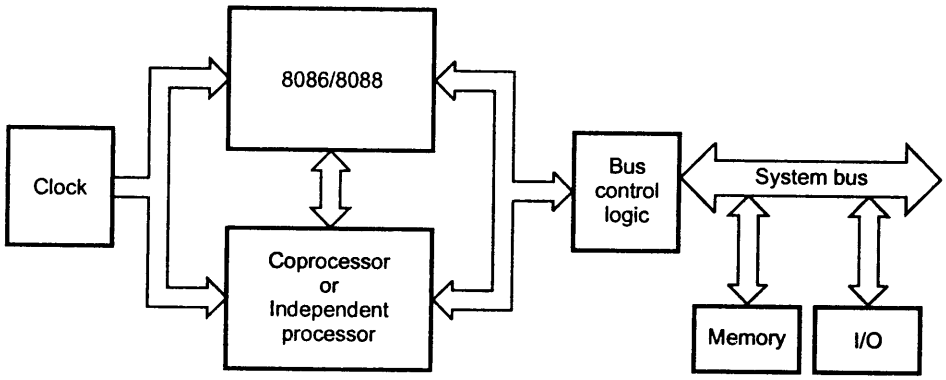


Fig. 11.2 Closely coupled configuration

Fig. 11.3 shows the interaction between CPU and independent processor in closely coupled configuration.

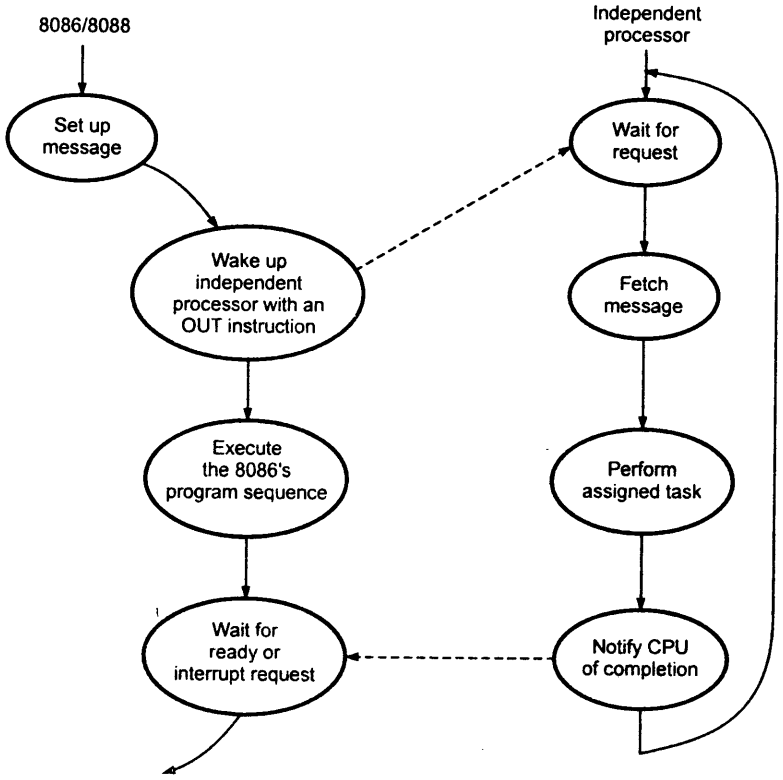


Fig. 11.3 Interaction between CPU and independent processor

In a closely coupled system no special instruction such as WAIT or ESC is used. Here the communication between host and independent processor is done through memory space. As shown in the Fig. 11.3, the host sets up a message in memory and wakes up

independent processor by sending command to one of its ports. The independent processor then accesses the memory to execute the task in parallel with the host. When task is completed, the external processor informs the host processor about the completion of task by using either a status bit or an interrupt request.

11.3 Loosely Coupled System using 8086

Fig. 11.4 shows the loosely coupled configuration. It consists of different modules. Each module may consists of an 8086, an another processor capable of being a bus master,

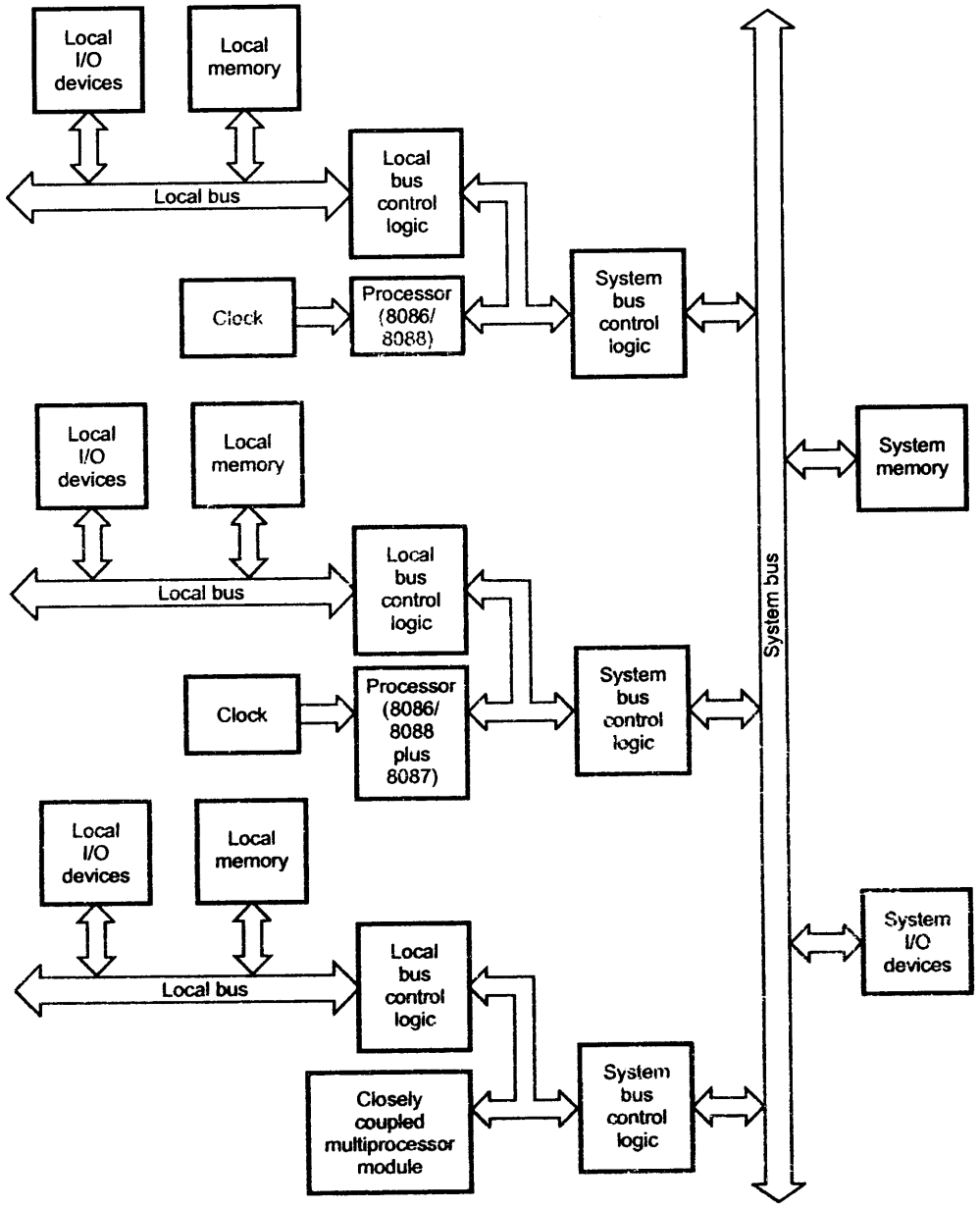


Fig. 11.4 Loosely coupled configuration

or a coprocessor or closely coupled configuration. Normally each processor has its own local memory and I/O devices, to which other processors do not have direct access. But they can share system resources.

Advantages of Loosely Coupled System

1. Better system throughput by having more than one processor.
2. Each processor may have a local bus to access local memory or I/O devices so that a greater degree of parallel processing can be achieved.
3. System structure is more flexible. As the system consists of different modules, one can easily add or remove modules to change the system configuration ; without affecting the other modules in the system.
4. A failure in one module normally does not cause a breakdown of the entire system. The faulty module can be detected and replaced.

11.4 The 8087 Numeric Data Processor

As mentioned earlier, multiprocessor system consists of processors and coprocessors. The numeric processor 8087 is a coprocessor which has been specially designed to work under the control of the processor 8086 and to support additional numeric processing capabilities.

11.4.1 Features of 8087

1. It can operate on data of the integer, decimal, and real types, with lengths ranging from 2 to 10 bytes.
2. Its instruction set not only includes various forms of addition and subtraction, but also provides many useful functions such as square root, exponential, tangent, and so on.
3. It is high performance numeric data processor. It can multiply two 64-bit real numbers in about 27 μs and calculate square root in about 36 μs.
4. It follows IEEE floating point standard.
5. It is multibus compatible.

11.4.2 Pin Diagram of 8087

Fig. 11.5 shows pin diagram of 8087. The address/data, status, ready, reset, clock, power and ground pins of the NDP are similar to the 8086 pins. Among the remaining 8 pins, four are not used. The other pins are as follows :

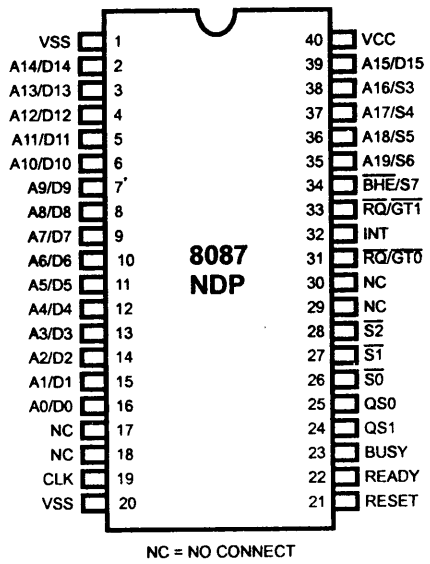


Fig. 11.5 Pin diagram of NDP 8087

1. $\overline{\text{BUSY}}$: $\overline{\text{BUSY}}$ signal from the 8087 is connected to the $\overline{\text{TEST}}$ input of the 8086. If the 8086 needs the result of some computation that the 8087 is doing before it can execute next instruction in the program, user can tell 8086 with a WAIT instruction to keep looking at its $\overline{\text{TEST}}$ pin until it finds the pin low. A low on the 8087 $\overline{\text{BUSY}}$ output indicates that the 8087 has completed the computation.

2. $\overline{\text{RQ}}$ / $\overline{\text{GT}}_0$: This request / grant signal from the 8087 is usually connected to the request /grant ($\overline{\text{RQ}}$ / $\overline{\text{GT}}_0$ or $\overline{\text{RQ}}$ / $\overline{\text{GT}}_1$) pin of the 8086.

3. $\overline{\text{RQ}}$ / $\overline{\text{GT}}_1$: This request/grant signal is connected to the request/grant pin of the independent processor such as 8089.

4. INT : The interrupt pin is connected to the interrupt management logic. The 8087 can interrupt the 8086 through this interrupt management logic at the time, error condition exists.

5. $\overline{S}_0$ - $\overline{S}_2$: These are the status bits of 8087 which are encoded as follows :

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Status
0	X	X	Unused
1	0	0	Unused
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

6. QS_0 - QS_1 : These signals give the queue status as follows :

QS_1	QS_0	Operation
0	1	No operation
0	1	First byte of opcode from Queue
1	0	Queue empty
1	1	Subsequent byte from the queue

11.4.3 Circuit Connection for 8087

Fig. 11.6 shows the circuit connection for 8087 with 8086. First note that the MN/$\overline{MX}$ pin of the 8086 is grounded, so the 8086 is operating in its maximum mode. Remember that in maximum mode the 8086 sends encoded control signals on the status lines $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ and the queue status lines QS_1 and QS_0 instead of generating the control signals directly. As shown in the Fig. 11.6, in maximum mode system an external controller such as the 8288 decodes these status signals to produce the control signals. These status signals also go directly from the 8086 to the 8087.

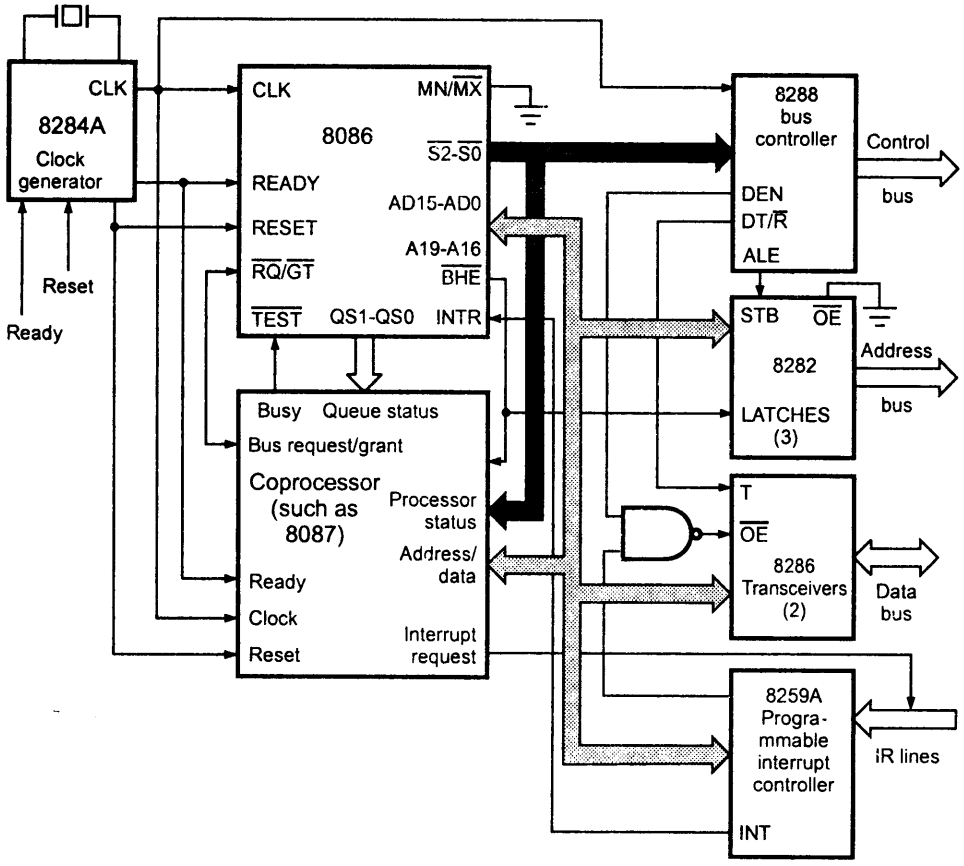


Fig. 11.6 Interconnection of 8086 and 8087

The upper address lines A_{16}-A_{19} are also connected directly from 8086 to the 8087. The 8087 receives the same clock and reset signals. The bus request/grant signal from 8087 is connected to $\overline{RQ}_0/\overline{GT}_0$ or $\overline{RQ}_1/\overline{GT}_1$ signal of the 8086. The Busy signal from 8087 is connected to the $\overline{TEST}$ input of the 8086. If the 8086 must have result of some computation that the 8087 is doing before it can execute its further instructions, we tell the 8086 with a WAIT instruction to keep looking at its $\overline{TEST}$ pin until it finds the pin low. A low on the $\overline{TEST}$ pin, i.e. on the BUSY output indicates that the 8087 has completed the computation.

11.4.4 Interaction between 8086 and 8087

The 8087 is an actual processor. It has its own specialized instruction set. Instructions for 8087 are inserted in the 8086 program as needed. Both 8086 and 8087 execute their instructions from the same program. As the 8086 fetches instruction bytes from memory and puts them in its internal queue, the 8087 also reads these instruction bytes and puts them in its internal queue. The 8087 decodes each instruction that comes into its queue. When it decodes an instruction from its queue and finds that it is an 8086 instruction, the 8087 simply treats the instruction as a NOP. Likewise, when 8086 decodes an instruction from its queue and finds that it is an 8087 instruction, the 8086 simply treats the instruction as a NOP, or in some cases reads one additional word from memory for 8087. Here, it is important to note that each processor decodes all of the instructions in the fetched instruction byte stream, but only executes its own instructions. The 8086 and 8087 instructions are differentiated by code 11011. The 8087 instruction code have 11011 as the most significant bits of their first code byte.

An instruction to be executed by 8087 is indicated when an ESC instruction appears in the program. The 8087 can keep a track for ESC instruction by monitoring the host 8086 $\overline{S}_2$ to $\overline{S}_0$ and AD_0-AD_{15} of 8086. The 8087 must track the instruction by monitoring the Q status QS_0-QS_1. If the Q status is 00, 8087 does nothing. If it is 01 it compares the five MSB bits with 11011. If there is a match, then an ESC instruction is fetched and executed by both, and 8087 will perform the indicated operation; otherwise, this byte is ignored and deleted from the queue. If an error occurs at the time of decoding an ESC instruction, 8087 sends an interrupt request.

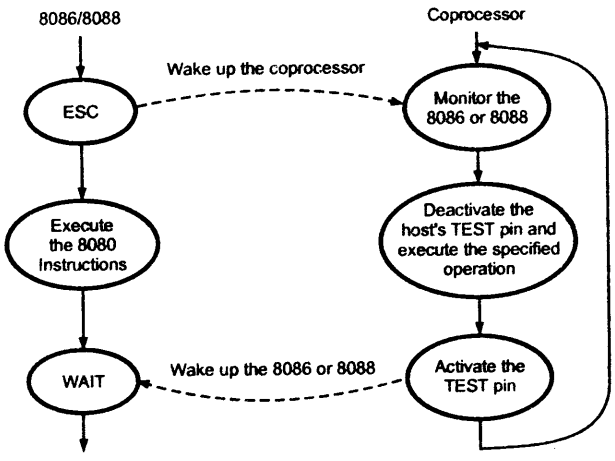


Fig. 11.7 Interaction between CPU and COP

When 8086 reads an 8087 instruction that needs data from memory or wants to send data to memory, the 8086 sends the memory address coded in the instruction and sends the appropriate memory-read or memory-write signals to transfer a word of data. In case of a memory-read, memory put the addressed data on the data bus. The 8087 then simply reads this word of the data bus and executes its instruction. The 8086 ignores the data word. Many times 8087 needs more than one word. In such situations, the 8086 outputs the address of the first data word on the address bus and outputs the appropriate memory-read or memory-write control signal. The 8087 reads the data word put on the data bus by memory or writes a data word to memory on the data bus. The 8087 then grabs the 20-bit physical address that was send by the 8086. To transfer the additional words, the 8087 then takes over the buses from the 8086. To take over

the bus the 8087 sends a low-going pulse on its $\overline{RQ}/\overline{GT}_0$ pin. The 8086 responds to this by sending another low going pulse back to the $\overline{RQ}/\overline{GT}_0$ pin of the 8087 and by floating its buses. The 8087 then increments the address it grabbed during the first transfer and outputs the incremented address on the address bus. When the 8087 outputs a memory-read or memory-write signal, another data word will be transferred to or from the 8087. The 8087 continues this process until it receives/sends all the data words required by the instruction to or from memory. When 8087 finishes its data transfer, it sends another low-going pulse on $\overline{RQ}/\overline{GT}_0$ pin to let the 8086 know it can have the buses back again.

While the 8087 is executing an instruction it asserts BUSY pin high. When it completes its instruction it drops its BUSY pin low. Since the BUSY pin from 8087 is connected to the $\overline{TEST}$ pin of the 8086, the 8086 can check this pin to see if 8087 has executed its instruction. The 8086 checks $\overline{TEST}$ pin with the help of WAIT instruction.

11.4.5 The 8087 Architecture

The internal architecture of 8087 is as shown in Fig. 11.8.

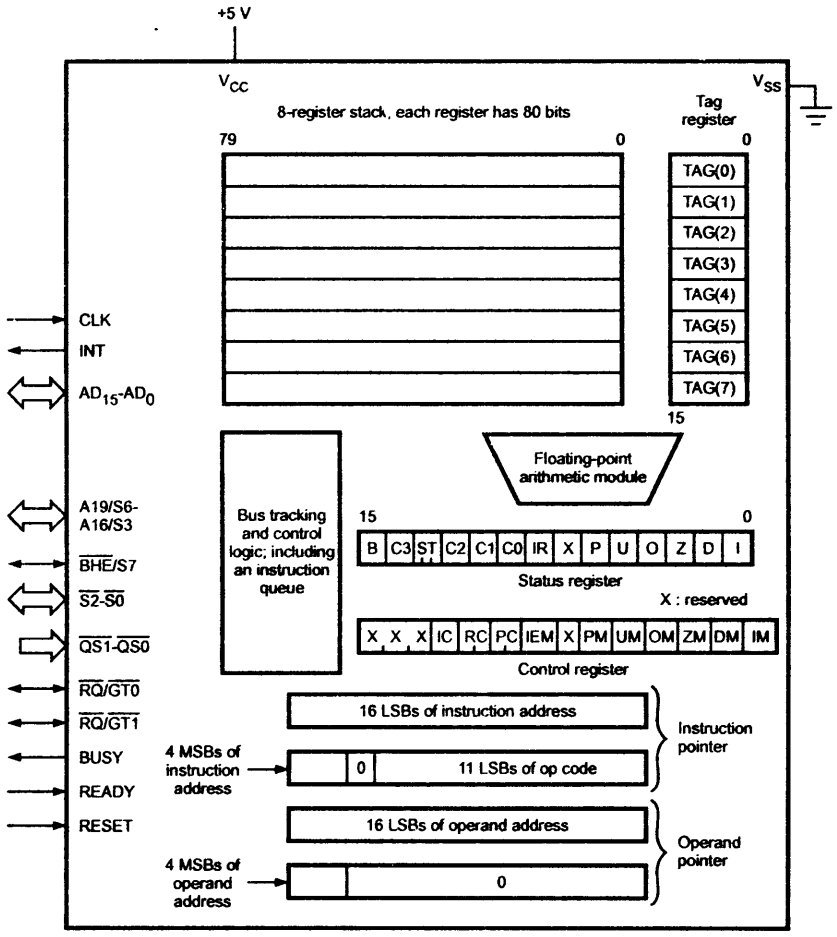


Fig. 11.8 Block diagram of 8087

11.4.5.1 Instruction Queue

It maintains a 6 byte instruction queue and tracks a execution sequence of the host. If the current host instruction is an ESC instruction, the 8087 decodes the external opcode to perform the specified operation and captures the operand address. The other instructions are ignored by 8087.

11.4.5.2 Data Registers

It has 8 data registers. Each register is 80-bit and it is accessed as a stack. An operand may be pushed or popped from top of stack. A 'push' operation decrements TOP of stack by 1 and loads a value into the new top register. A 'pop' operation stores the value from the current top register and then increments TOP by 1. The top stack element is pointed by ST bits, i.e., bits 13,12 and 11 of the status register.

11.4.5.3 Status Registers

The status register is 16-bit register. It reflects the overall state of the 8087. It indicates various errors, stores condition code for certain instructions, specifies which register is top of the stack and indicates the $\overline{\text{BUSY}}$ status. Fig. 11.9 shows the bit definitions of the Status Register.

Error Flags

IE : An invalid operation such as stack overflow, stack underflow, invalid operand, square root of a negative number etc.

DE : The operand is not normalized.

ZE : A divide by zero error.

OE : An exponent overflow error, i.e., the biased exponent is too small.

PE : A precision error, i.e., the result cannot be represented in the designated format and hence is rounded off.

Interrupt Flag

IR : Indicates the existence of the interrupt request.

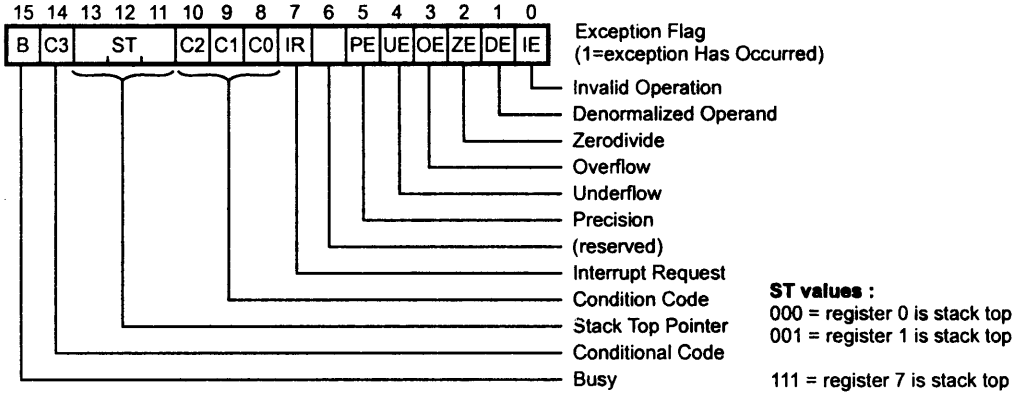


Fig. 11.9 Bit pattern of status register

Condition Code

C_0 - C_3 indicate the condition code. The condition code is set by the compare and examine instructions, which are discussed later.

Stack Bits

ST : $\overline{S}_0$ - $\overline{S}_2$ indicate the top of stack.

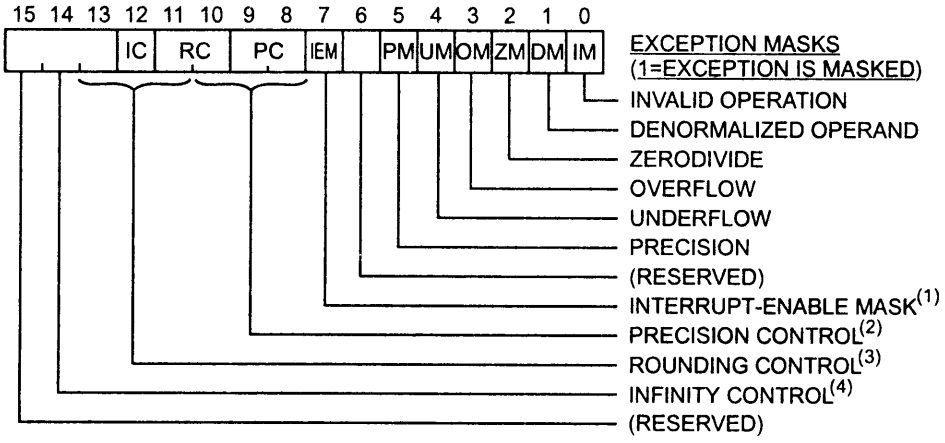
Busy Status

B : Indicates current operation is not completed.

NOTE : After the 8087 is reset or initialized, all status bits except the condition code are cleared.

11.4.5.4 Control Register

The control register is also 16-bit. The 8087 provides several processing options which are selected by loading a word from memory into the control register. The control register gives the facility to mask each error type individually from causing an interrupt. It can be used to set precision levels, rounding type and infinity representation. Fig. 11.10 shows the bit definition of control register.



(1) Interrupt - Enable Mask
0 = Interrupts Enable
1 = Interrupts Disabled (Masked)
(2) Precision Control
00 = 24 bits
01 = (reserved)
10 = 53 bits
11 = 64 bits

(3) Rounding Control
00 = Round to Nearest or Even
01 = Round Down (toward %)
10 = Round Up (toward %)
11 = Chop (Truncate Toward Zero)
(4) Infinity Control
0 = Projective
1 = Affine
DEFAULT AFTER FINITE

Fig. 11.10 Bit pattern of control register

The low order byte of this control register configures 8087 interrupts and exception masking. Bits 5-0 of the control register contain individuals masks for each of the six exceptions that the 8087 recognizes and bit 7 contains a general mask bit for all 8087 interrupts. The high order byte of the control register configures the 8087 operating mode including precision, rounding, and infinity controls.

After reset or initialization of the 8087, these bits are PC = 11, RC = 00, IC = 0, IEM = 0 and all error mask bits are 1.

Tag Register : TAG register holds the status of the contents of data register. This includes

0 0 - Data Valid

0 1 - Zero

1 0 - A special value i.e. NAN (Not A Number), infinity or denormal.

1 1 - Empty

11.4.6 Data Formats and Conversions

The 8087 can operate on memory operands of seven different data types :

- Word integer
- Short integer
- Long integer
- Packed BCD
- Short real
- Long real
- Temporary real

Fig. 11.11 shows the number of bytes, format, and approximate range for each of these data types. In memory, the least significant byte of a number is always stored at the lowest address. The signed integer numbers or BCD numbers are referred to as **fixed point numbers** because they don't have any information about decimal point in the number. The decimal point is always assumed to be to the right of the least significant digit, so all numbers represented in this form are whole numbers with no fractional part. To represent fractional numbers in this format, programmer has to keep a track of decimal point. But it is possible to represent number in the format which has the integer part and the fractional part. This format automatically keeps a track of the position of decimal number. The number represented in such formats are called as **real numbers** or **floating point numbers.**

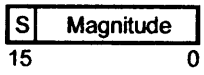
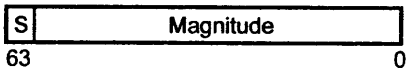
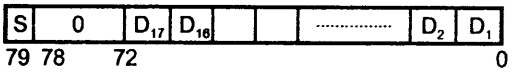
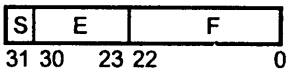
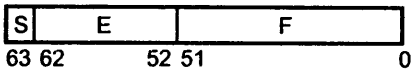
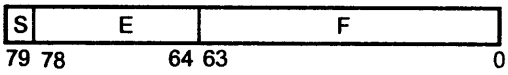
Data type	Bytes	Approximate Range (decimal)	Format
Word integer	2	–32,768 to 32,767	
Short integer	4	$\approx -2 \times 10^{9}$ to 2×10^{9}	
Long integer	8	$\approx -9 \times 10^{18}$ to 9×10^{18}	
Packed BCD	10	$-10^{18} + 1$ to $10^{18} - 1$	
Short real	4	$\approx \pm 1 \times 10^{-38}$ to $\pm 3 \times 10^{38}$	
Long real	8	$\approx \pm 10^{-308}$ to $\pm 10^{308}$	
Temporary real	10	$\approx \pm 10^{-4932}$ to $\pm 10^{4932}$	

Fig. 11.11 Data formats of NDP 8087

A real format is divided into three fields :

1. Sign
2. Exponent
3. Mantissa.

Real number $n = \text{sign}\ 2^{exp} \times \text{mantissa}$.

To convert any number to real format, we have to move the decimal point to the right of the most significant, nonzero digit. Then it is necessary to multiply number with base N. N represents the number of digits moved to adjust decimal point. This process of moving the decimal point to the right of the most significant, nonzero digit is referred to as **Normalization.**

The 8087 recognizes three real data types :

- Short real (32-bit)
- Long real (64-bit)
- Temporary real (80-bits)

As mentioned earlier real format has field : sign, exponent and mantissa.

In 8087 bias value is added to the true exponent. This solves the problem of representation of negative exponent and hence the magnitude of two numbers can be compared without having to do arithmetic on the exponents first. The sign bit is '0' for positive numbers and '1' for negative numbers. To make this clear, we will see few examples :

Example 11.1 : *Convert* 1259.125_{10} *into short real, long real and temporary real format.*

Solution : Step 1 : Convert this decimal number to binary format.

Integer part :

		Remainders	
16	1259	B	LSB
16	78	E	
	4	4	

MSB

$$4EB_H = \frac{(100}{4}\ \frac{1110}{E}\ \frac{1011)_2}{B}$$

Fractional part :

$.125 \times 2 = 0.25$	0
$0.25 \times 2 = 0.5$	0
$0.5 \times 2 = 1.0$	1
0	

$= .001$

Binary number = 10011101011 + .001 = 10011101011.001

Step 2 : Normalize the number.

$10011101011.001 = 1.0011101011001 \times 2^{10}$

Now we will see the representation of the number in different formats.

1. Short real

For the given number

$S = 0, \quad E = 10, \quad F = 0011101011001$

Bias for the short real format = 127

$\therefore$ Biased exponent = 10 + 127 = 137 = 10001001

Number in the short real format

0 10001001 0011101011001 ... 0 = 1259.125_{10}

S Exponent Fraction

2. Long real

Bias for long real format = 1023

$\therefore$ Biased exponent = 10 + 1023 = 1033 = 10000001001

Number in long real format

0 10000001001 0011101011001 ... 0 = 1259.125_{10}

S Exponent Fraction

3. Temporary real format

Bias for temporary real format = 16383

$\therefore$ Biased exponent = 10 + 16383 = 16393 = 100 0000 0000 1001

Number in temporary real format

0 100 0000 0000 1001 001110101001 ... 0 = 1259.125_{10}

S Exponent Fraction

Example 11.2 : *Convert* - 307.1875_{10} *into short real, long real and temporary real format.*

Solution : Step 1 : Convert decimal number in binary format.

Integer part : Convert this decimal number to binary format.

		Remainders	
16	307	3	LSB
16	19	3	
	1		

MSB

$$= \frac{1}{1}\frac{0011}{3}\frac{0011}{3}$$

Fractional part :

$$\begin{array}{l|llll} .1875 & \times & 2 & = 0.3750 & 0 \\ .375 & \times & 2 & = 0.750 & 0 \\ .750 & \times & 2 & = 1.5 & 1 \\ .5 & \times & 2 & = 1.0 & 1 \end{array}$$

$$= .0011$$

$$\text{Binary number} = -100110011 + .0011$$

$$= -100110011.0011$$

Step 2 : Normalize the number.

$$-100110011.0011 = -1.001100110011 \times 2^8$$

Now we will see the representation of the number in different formats.

1. Short real

For the given number

$$S = 1 \quad E = 8 \quad F = 001100110011$$

Bias for short real format = 127

$\therefore$ Biased exponent = 8 + 127 = 135 = 10000111

Number in short real format

1 10000111 00110011000 ... 0 = -307.1875_{10}

S Exponent Fraction

2. Long real

Bias for long real format = 1023

$\therefore$ Biased exponent = 8 + 1023 = 1031 = 10000000111

Number in long real format

1 10000000111 0011001100110 ... 0 = -307.1875_{10}

S Exponent Fraction

3. Temporary real

Bias for temporary real format = 16383

$\therefore$ Biased exponent = 8 + 16383 = 16391 = 100000000000111

Number in temporary real format

1 100000000000111 0011001100110 ... 0 = -307.1875_{10}

S Exponent Fraction

Example 11.3 : *Convert* 0.0625_{10} *into short real, long real and temporary real format.*

Solution : Step 1 : Convert this decimal number to binary format.

Integer part : 0

Fractional part :

0.0625	×	=	2	0.125	0
0.125	×	=	2	0.25	0
0.25	×	=	2	0.5	0
0.5	×	=	2	1.0	1

= 0.0001

Binary number = 0 + .0001 = 0.0001

Step 2 : Normalize the number.

$0.0001 = 1.0 \times 10^{-4}$

Now we will see the representation of the number in different formats.

1. Short real

For the given number

S = 0 E = − 4 F = 0000

Bias for the short real number = 127

$\therefore$ Biased Exponent = − 4 + 127 = 123 = 01111011

Number in the short real format

0 01111011 00000 ... 0 = 0.0625_{10}

S Exponent Fraction

2. Long real

Bias for the long real format = 1023

$\therefore$ Biased Exponent = − 4 + 1023 = 1019 = 01111111011

Number in the long real format

0 01111111011 0000 ... 0 = 0.00625_{10}

S Exponent Fraction

3. Temporary real

Bias for the temporary real format = 16383

$\therefore$ Biased exponent = − 4 + 16383 = 16379 = 011111111111011

Number in the temporary real format

0	01111111111011	0000 ... 0
S	Exponent	Fraction

NOTE :

1. A biased exponent with all 1's is reserved to represent infinity or "not-a-number" (NAN). At the other extreme, a biased exponent with all 0's is reserved to represent + 0 (all 0s with a sign bit 0).
2. – 0 (all 0s with a sign bit 1), is a denormal. A denormal is a result that causes an underflow and has leading 0s in the mantissa even after the exponent is adjusted to its smallest possible value. NANs and denormals are usually used to indicate overflows and underflows, respectively.
3. The 8087 works internally with all numbers in the 80-bit temporary real format. (1 bit for sign, 15-bits for exponent and 64-bits for the mantissa). The temporary real format is used to reduce the chances for overflows and underflows during a series of calculations which produce a final result within the desired range.

11.4.7 Stacks in 8087

The 8087 has a 3-bit stack pointer which holds the number of the register which is the current top-of-stack. When the 8087 is initialized, the 3-bit stack pointer in the 8087 is loaded with 000, that indicates register 0 is a top of stack.

As shown in Fig. 11.12, the stack of 8087 is circular. So if you decrement 000 you get 111. When 8087 reads the first number, stack is decremented to 111(7) and the number is stored in register number 111(7). Now register 7 is the top of stack.

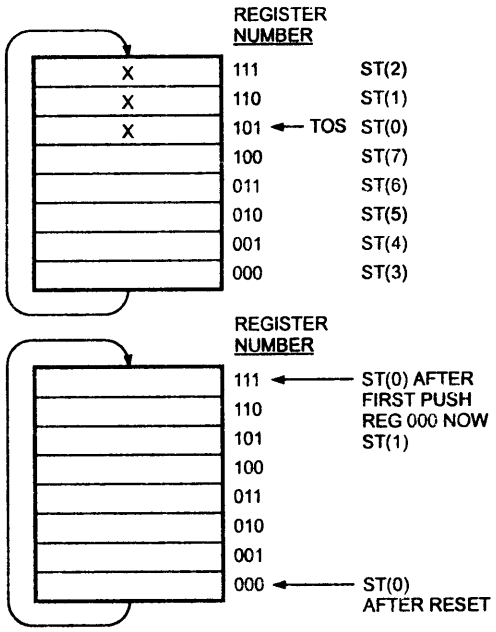


Fig. 11.12 Register stack in 8087

In the 8087 instructions, the register that is currently top of stack is referred to as ST(0), or simply ST and register next to it is referred to as ST(1). The register 'next to it' means the stack pointer will point that register if we pop one number from stack.

11.4.8 Instructions of 8087

The 8087 instructions can be distinguished from 8086 instructions by letter F which stands for floating point number. All mnemonics in 8087 begins with letter F. In all 8087 has 68 instructions, which can be divided into six groups.

- Data transfer instructions
- Arithmetic instructions
- Compare instructions
- Transcendental instructions
- Load constant instructions
- Processor control instructions

11.4.8.1 Data Transfer Instructions

a) Real Transfers

FLD source : Decrements the stack pointer by one and copies a real number from a stack element or memory location to the new ST. A short-real or long-real number from memory is automatically converted to temporary real format by the 8087 before it is put in ST.

Exceptions : I, D.

Examples :

```
FLD ST(2)   ; Copies ST(2) to ST
FLD [BX]    ; Number from memory pointed by BX copied to ST
```

FST destination : Copies ST to a specified stack position or to a specified memory location.

Exceptions : I, O, U, P.

Examples :

```
FST ST(3)   ; Copy ST to ST(3)
FST [BX]    ; Copy ST to memory pointed by [BX]
```

FSTP destination : Copies ST to a specified stack element or memory location and increments the stack pointer by one to point to the next element on the stack. This is a stack POP operation.

FXCH destination : Exchanges the contents of ST with the contents of a specified stack element. If no destination is specified, then ST(1) is used.

Exception : I.

Example :

```
FXCH ST(4)          ; Swap ST and ST(4)
```

b) Integer Transfers

FILD source : Integer load. Converts integer number from memory to temporary-real format and pushes converted number on 8087 stack.

Exception : I.

Example :

```
FILD DWORD PTR [BX]    ; Short integer from memory location pointed
                       ; by [BX]
```

FIST destination : Integer store. Converts number from ST to integer form, and copies to memory.

Exceptions : I, P

Example :

```
FIST INT_NUM        ; ST to memory locations named INT_NUM
```

FISTP destination : Integer store and pop. Similar to FIST except that stack pointer is incremented after copy.

c) Packed Decimal Transfers

FBLD source : Packed decimal (BCD) load. Convert number from memory to temporary-real format and push on top of 8087 stack.

Exception : I.

Example :

```
FBLD AMOUNT  ; Ten byte BCD number from
             ; memory location AMOUNT to ST
```

FBSTP destination : BCD store in memory and pop 8087 stack. Pops temporary-real from stack, converts to 10-byte BCD, and stores result to memory.

Exception : I.

Example :

```
FBSTP MONEY  ; Contents from top of stack are converted to BCD,
             ; and stored in memory
```

11.4.8.2 Arithmetic Instructions

a) Addition

FADD destination, source : Adds real number from specified source to real number at specified destination. Source can be stack element or memory location. Destination must be a stack element. If no source or destination is specified, then ST is added to ST(1) and the stack pointer is incremented so that the result of the addition is at ST.

Exceptions : I, D, O, U, P.

Examples :

```
FADD ST(2), ST      ; Add ST to ST(2), result in ST(2)
FADD ST, ST(5)      ; Add ST(5) to ST, result in ST
FADD SUM            ; Real number from memory + ST
FADD                ; ST + ST(1), pop stack-result at ST
```

FADDP destination, source : Adds ST to specified stack element and increments stack pointer by one.

Exceptions : I, D, O, U, P.

Example :

```
FADDP ST(2)         ; Add ST(2) to ST.
                    ; Increment stack pointer so ST(2) becomes ST.
```

FIADD source : Adds integer from memory to ST, stores the result in ST.

Exceptions : I, D, O, P.

Example :

```
FIADD CARS_SOLD     ; Integer number from memory + ST
```

b) Subtraction

FSUB destination, source : Subtracts the real number at the specified source from the real number at the specified destination and puts the result in the specified destination.

Exceptions : I, D, O, U, P.

Examples :

```
FSUB ST(3), ST      ; ST(3) ← ST(2) - ST
FSUB DIFFERENCE     ; ST ← ST - real from memory
FSUB                ; ST ← (ST(1) - ST)
```

FSUBP destination, source : Subtracts ST from specified stack element and puts result in specified stack element. Then increments stack pointer by one.

Exceptions : I, D, O, U, P.

Examples :

```
FSUBP ST(2)     ; ST(2) – ST . ST(1) becomes new ST.
```

FISUB source : Subtracts integer number stored in memory from ST and stores result in ST.

Exceptions : I, D, O, P.

Example :

```
FISUB DIFFERENCE    ; ST ← ST - integer
                    ; from memory
```

c) Reversed Subtraction

FSUBR destination, source

FSUBRP destination, source

FISUBR source

These instructions operate same as the FSUB instructions described above except that these instructions subtract the contents of the specified destination from the contents of the specified source and put the difference in the specified destination.

NOTE : Normal FSUB instruction subtracts source from destination.

d) Multiplication

FMUL destination, source : Multiply real number from source by real number from specified destination, and put result in specified stack element. Exceptions : I, D, O, U, P.

```
FMUL ST(2), ST      ; Multiply ST(2) and ST, result in ST(2)
FMUL ST, ST(5)      ; Multiply ST(5) to ST, result in ST
```

FMULP destination, source-Multiplies real number from specified source by real number from specified destination, puts result in specified stack element, and increment stack pointer by one. With no specified operands FMULP multiplies ST(1) by ST and pops stack to leave result at ST.

Exceptions : I, D, O, U, P.

```
Example : FMULP ST(2)       ; Multiply ST(2) to ST. Increment stack
                            ; pointer so STI (1) becomes ST
```

FIMUL source : Multiply integer from memory ST and put result in ST.

Exceptions : I, D, O, P.

Example :

```
FIMUL DWORD PTR [BX]    ; integer number from memory pointed by BX × ST
                        ; and result in ST
```

e) Division

FDIV destination, source : Divides destination real by source real, stores result in destination.

Exceptions : I, D, Z, O, U, P.

Example :

```
FDIV ST(2), ST          ; Divides ST by ST(2)
                        ; stores result in ST
```

FDIVP destination, source : Same as FDIV, but also increments stack pointer by one after DIV.

Exceptions : I, D, Z, O, U, P.

Example :

```
FDIV ST(2), ST    ; Divides ST by ST(2), stores result in ST
                  ; and increments stack pointer
```

FIDIV source : Divides ST by integer from memory, stores result in ST.

Exceptions : I, D, Z, O, U, P.

Example :

```
FIDIV PERCENTAGE    ; ST ← ST/integer number
```

f) Reversed Division

FDIVR destination, source

FDIVP destination, source

FIDIVR source

These three instructions are identical in format to the FDIV, FDIVP, and FIDIV instructions above except that they divide the source operand by the destination operand and put the result in the destination.

g) Other Arithmetic Operations

FSQRT : Contents of ST are replaced with its square root.

Exceptions : I, D, P.

Example : FSQRT

FSCALE : Scales the number in ST by adding an integer value in ST(1) to the exponent of the number in ST. Fast way of multiplying by integral powers of two.

Exceptions : I, O, U.

FPREM : Partial remainder. The contents of ST(1) are subtracted from the contents of ST over and over again until the contents of ST are smaller than the contents of ST(1).

Exceptions : I, D, U.

Example : FPREM

FRNDINT : Round number in ST to an integer. The round-control (RC) bits in the control word determine how the number will be rounded.

Exceptions : I, P.

FXTRACT : Separates the exponent and the significant parts of a temporary- real number in ST. After the instruction executes, ST contains a temporary-real representation of the significant of the number and ST (1) contains a temporary-read representation of the exponent of the number.

Exception : I.

FABS : Replaces ST by its absolute value. Instruction simply makes sign positive.

Exception : I.

FCHS : Complements the sign of the number in ST.

Exception : I.

11.4.8.3 Compare Instructions

Compares the contents of ST with contents of specified or default source. The source may be another stack element or real number in memory. These compare instructions set the condition code bits C3, C2, and C0 of the status word shown in Table 11.1.

Table (a)		
Order	C3	C0
(ST) > (SRC)	0	0
(ST) < (SRC)	0	1
(ST) = (SRC)	1	0
Not comparable	1	1

Table (b)		
Order	C3	C0
(ST) > 0 . 0	0	0
(ST) < 0 . 0	0	1
(ST) = 0 . 0	1	0
Not comparable	1	1

Table (c)				
(ST)	C3	C2	C1	C0
+ Unnormal	0	0	0	0
+ NAN	0	0	0	1
– Unnormal	0	0	1	0
– NAN	0	0	1	1
+ Normal	0	1	0	0
+ ∞	0	1	0	1
– Normal	0	1	1	0
– ∞	0	1	1	1
+ 0	1	0	0	0
Empty	1	0	0	1
– 0	1	0	1	0
Empty	1	0	1	1
+ Denormal	1	1	0	0
Empty	1	1	0	1
– Denormal	1	1	1	0
Empty	1	1	1	1

Table 11.1

You can transfer the status word to memory with the 8087 FSTSW instruction and then use 8086 instructions to determine the results of the comparison.

Different compare instructions

FCOM source : Compares ST with real number in another stack element or memory.

Exceptions : I, D.

Examples :

```
FCOM          ; Compares ST with ST(1)
FCOM ST(4)    ; Compares ST with ST(4)
FCOM VALUE    ; Compares ST with real number
              ; from memory
```

FCOMP source : Identical to FCOM except that the stack pointer is incremented by one after the compare operation.

FCOMPP : Compares ST with ST(1) and increments stack pointer by 2 after compare.

Exceptions : I, D.

FICOM source : Compares ST to a short or long integer from memory.

Exceptions : I, D.

FICOMP source : Identical to FICOM except stack pointer is incremented by one after compare.

FTST : Compares ST with zero.

Exceptions : I, D.

FXAM : Tests ST to see if it is zero, infinity, unnormalized, or empty. Sets bits C_3, C_2, C_1 and C0 to indicate result. Refer Table 11.1.

Exceptions : None.

11.4.8.4 Transcendental (Trigonometric and Exponential) Instructions

FPTAN : Computes the values for a ratio of Y/X for an angle in ST. The angle must be expressed in radians, and the angle must be in the range of $0 < \text{angle} < \pi/4$.

NOTE : FPTAN does not work correctly for angles of exactly 0 and $\pi/4$. Exceptions : I, P.

FPATAN : Computes the angle whose tangent is Y/X. The X value must be in ST, and the Y value must be in ST(1). Also, X and Y must satisfy the inequality $0 < Y < X < \infty$. The resulting angle expressed in radians replaces Y in the stack. After the operation the stack pointer is incremented so the result is then ST. Exceptions : U, P.

F2XM1 : Computes the function $Y = 2^X - 1$ for an X value in ST. The result, Y, replaces X in ST. X must be in the range $0 \le X \le 0.5$

Exceptions : U, P.

FYL2X : Calculates Y times the log to the base 2 of X or $Y(\log_2 X)$. X must be in the range of $0 < X < \infty$ and Y must be in the range $-\infty < Y < +\infty$. X must initially be in ST and Y must be in ST(1). The result replaces Y and then the stack is popped so that the result is then at ST.

Exceptions : P

FYL2XP1 : Computes the function Y times the log to the base 2 of (X + 1) or $Y(\log_2 (X + 1))$. This instruction is almost identical to FYL2X except that it gives more accurate results when computing the log of a number very close to one.

11.4.8.5 Instructions which Load Constants

These instructions simply push the indicated constant onto the stack.

FLDZ - Push 0.0 onto stack
FLD1 - Push + 1.0 onto stack
FLDPI - Push the value of π onto stack
FLD2T - Push log of 10 to the base 2 onto stack ($\log_2 10$)
FLDL2E - Push log of e to the base 2 onto stack ($\log_2 e$)
FLDLG2 - Push log of 2 to the base 10 onto stack ($\log_{10} 2$)